Applicability of object-based storage devices in parallel file systems

Pete Wyckoff

Ohio Supercomputer Center

pw@osc.edu

HECURA Showcase 7 aug 07

Vision

- Processors faster, disk densities up, but I/O rates comparatively flat
- Leverage intelligent peripherals to improve scalability, managability, performance

Serverless parallel distributed file systems

Motivation

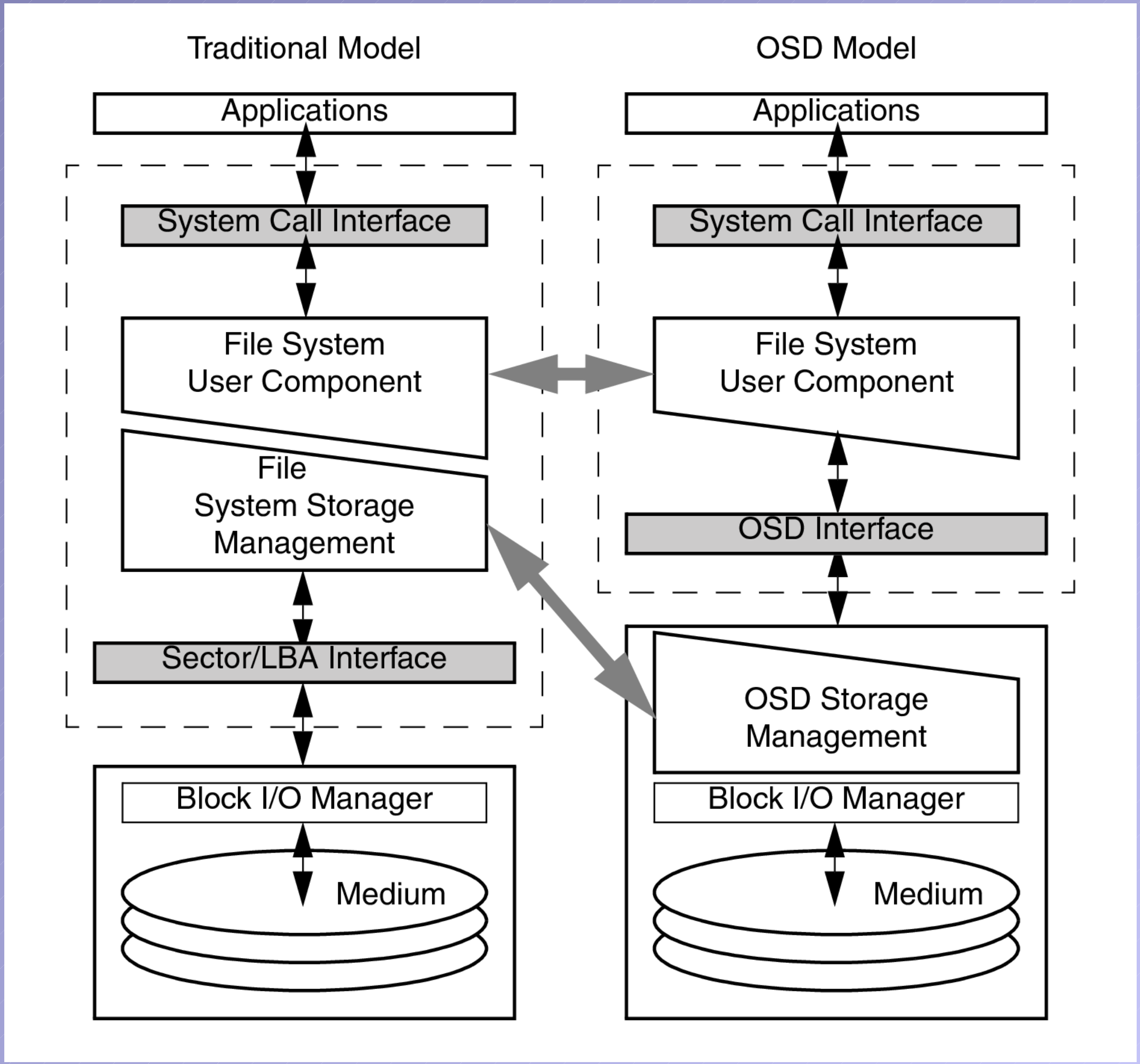
- OSDs offer higher-level semantic interface
- Secure, direct access of storage by clients

- Our work
  - Examine role of OSDs in parallel file systems
  - Analyze trade-offs of using OSDs for various aspects of parallel FSes
  - Develop extensions required for efficient use of OSDs in HPC parallel environments

OSD Background

- T10 specification
- Prototypes
- Emulators

- Pure target
- Security model



Mapping Data to Objects

- Block-based "objects" are 512 bytes
- OSD objects could be files
- For striped files, object is stripe, or stripe set?
- Collection feature allows grouping objects
  - fast searching
  - choose object boundaries to coincide with search
- Delegating object creation to clients
- When to create objects on create?
  - lazy, preallocate

Metadata

- OSDs store *attributes* with objects
- How do parallel FS attributes map to these?
  - uid, gid, perm, [acm]time, type
    - mtime of object vs mtime of file
  - data distribution: mapping of bytes to objects
  - POSIX extended attributes
- Managing collective behavior
  - object allocation, fsck, rebalancing
- Select objects by attribute
  - collections for indexing applied to metadata

Directories

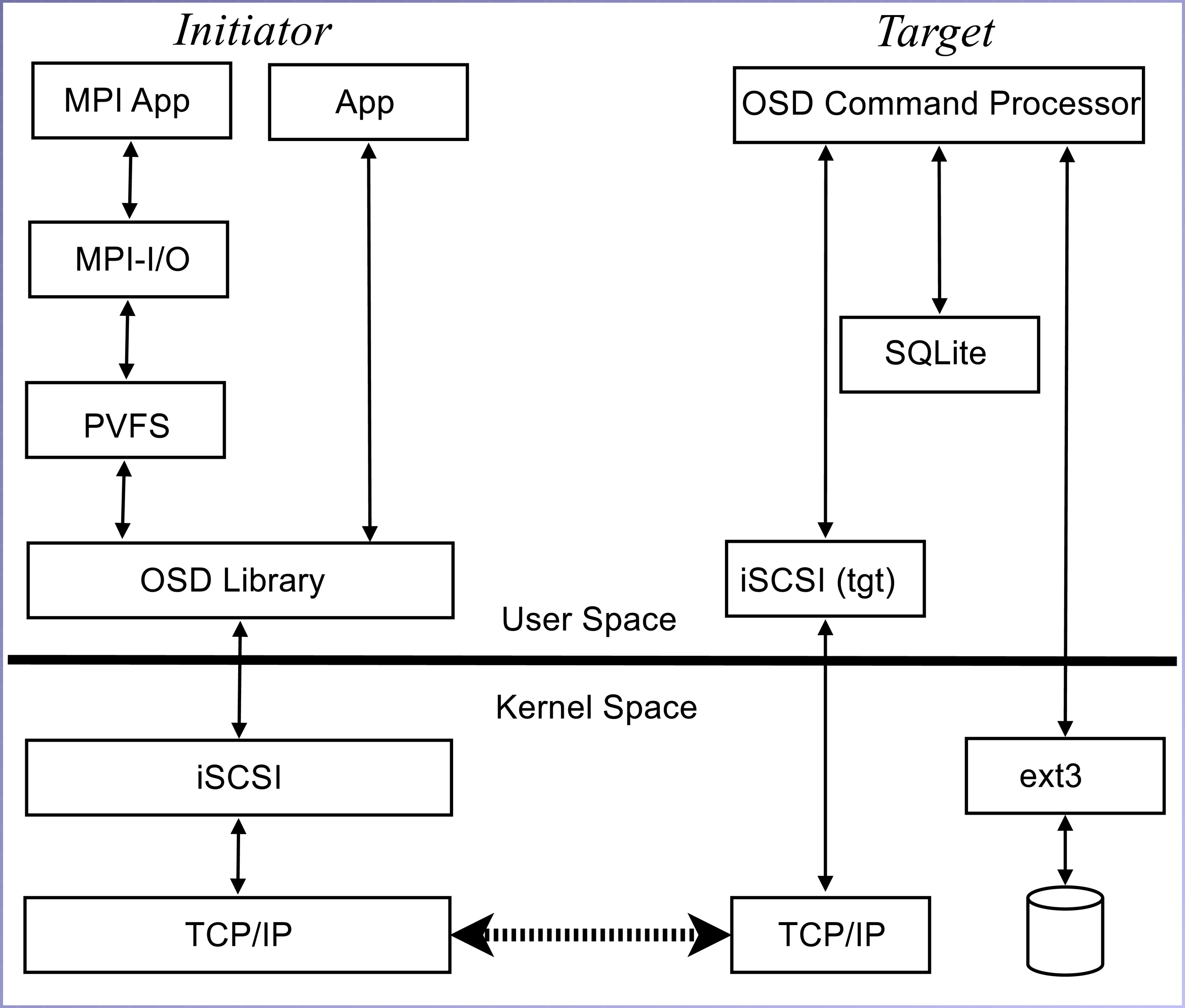
- Familiar form of data organization
- No consistency to relax for directories
- Need overall sequential behavior for any single directory
- For passive targets, this may be impossible

Progress

- Full stack for OSD work
  - initiator library
  - iSCSI target
  - OSD emulator

- Parallel file system using OSDs
  - ✓ Phase 1: datafile
    - striping, handle mapping, object management
  - ✓ Phase 2: metafile
    - handle mapping, metadata storage, distributions
  - Phase 3: directory
    - atomicity, tree layout, POSIX corner cases

Stack



Initiator
Target
MPI App
App
OSD Command Processor
MPI-I/O
SQLite
PVFS
OSD Library
iSCSI (tgt)
User Space
Kernel Space
iSCSI
ext3
TCP/IP
TCP/IP

Initiator

- Create CDBs
- Submit commands
- Retrieve results
- Generate and parse get/set attributes
- Device enumeration and mapping
- Python interface and basic unit tests
- Use "bsg" interface to SCSI midlayer (+ fixes)
  - extended CDBs
  - bidirectional
  - iovec
- Use "bidi" patches from Panasas (+ fixes)
- Why not Intel or IBM initiators?
  - reliance on old kernel APIs
  - non-bidirectional

Target

- Use "stgt" for SCSI and iSCSI
  - all userspace
  - leading approach for linux adoption
  - active development
- Many changes to stgt, most merged
  - OSD target command processing
  - iSCSI bidirectional implementation
  - iSER (iSCSI/RDMA) implementation
  - bug fixes
- Why not kernel-based iSCSI target?
  - not the linux way
  - living in out-of-tree modules is difficult

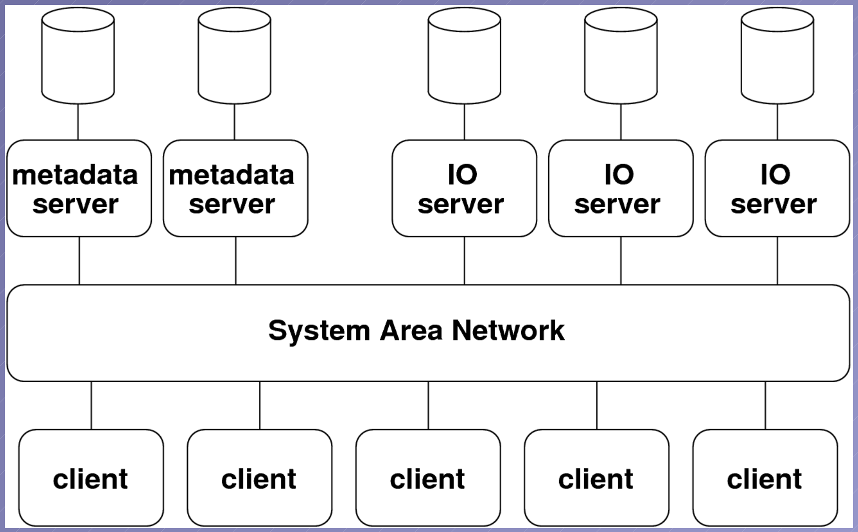
OSD Target Emulator

- Processes the commands delivered by stgt
- Command categories
  - object manipulation (done)
  - attribute manipulation (done)
  - I/O (done)
  - security (not done)
  - device management (partial)
  - collections (soon)
- Object storage
  - via POSIX and ext3: pread, pwrite
- Attributes
  - via SQLite, full SQL implementation
  - complex operations very succinct and fast

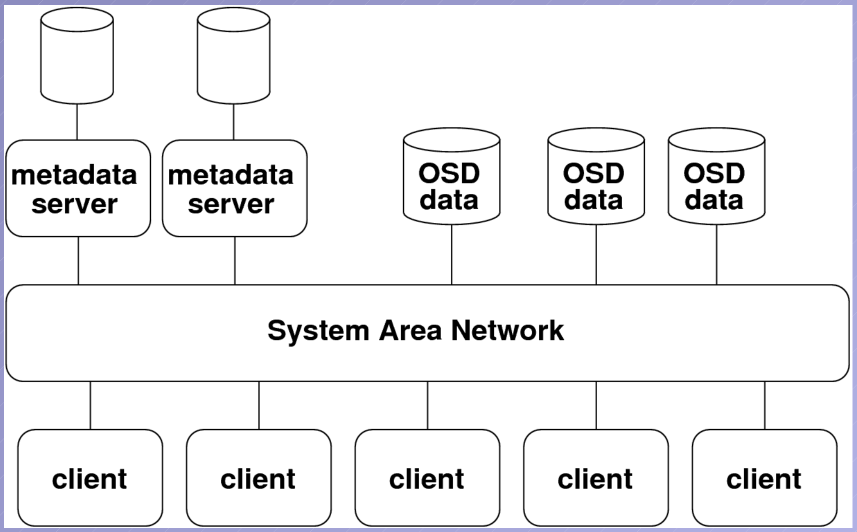
Other OSD Target Emulators

- IBM ObjectStone
  - binary-only x86-32 blob
  - cannot modify or evaluate
- Intel uosd
  - very simple, single-file implementation
  - only 8 basic commands
  - each attribute stored in a separate file
- Du, UMN
  - extensions to Intel
  - added extensive security infrastructure
  - same attribute implementation

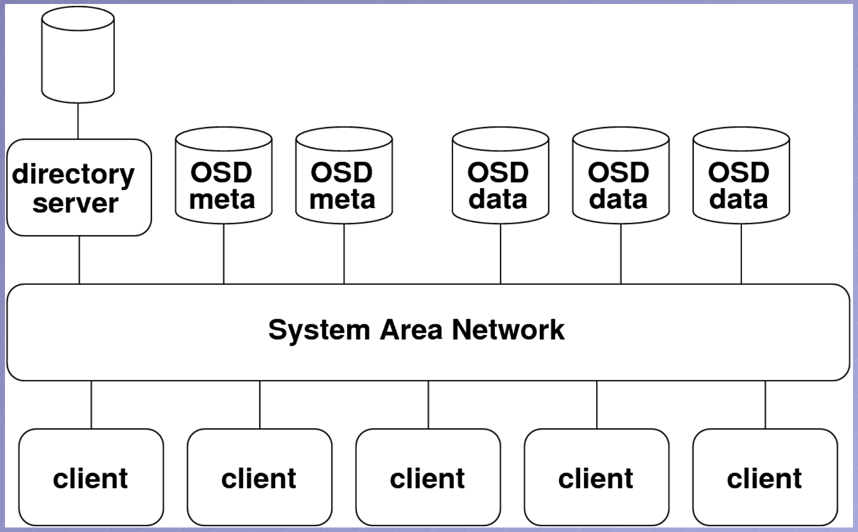
PVFS Modifications



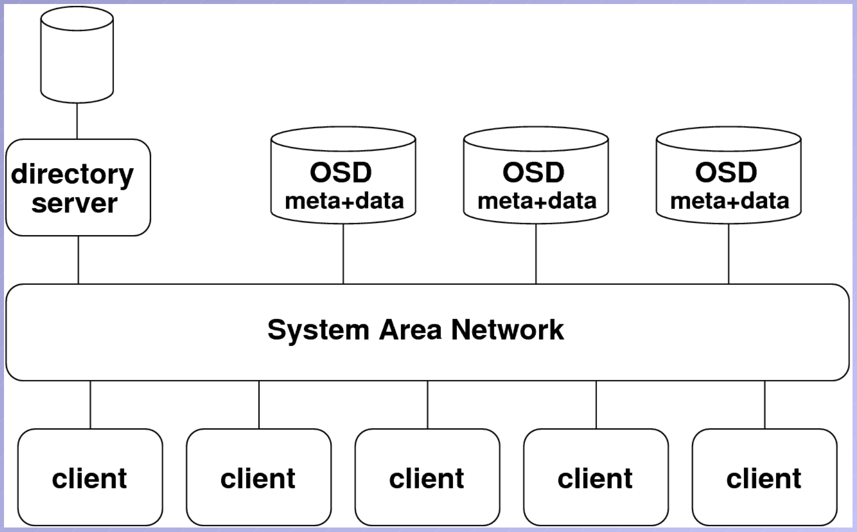
stock



datafile



metafile



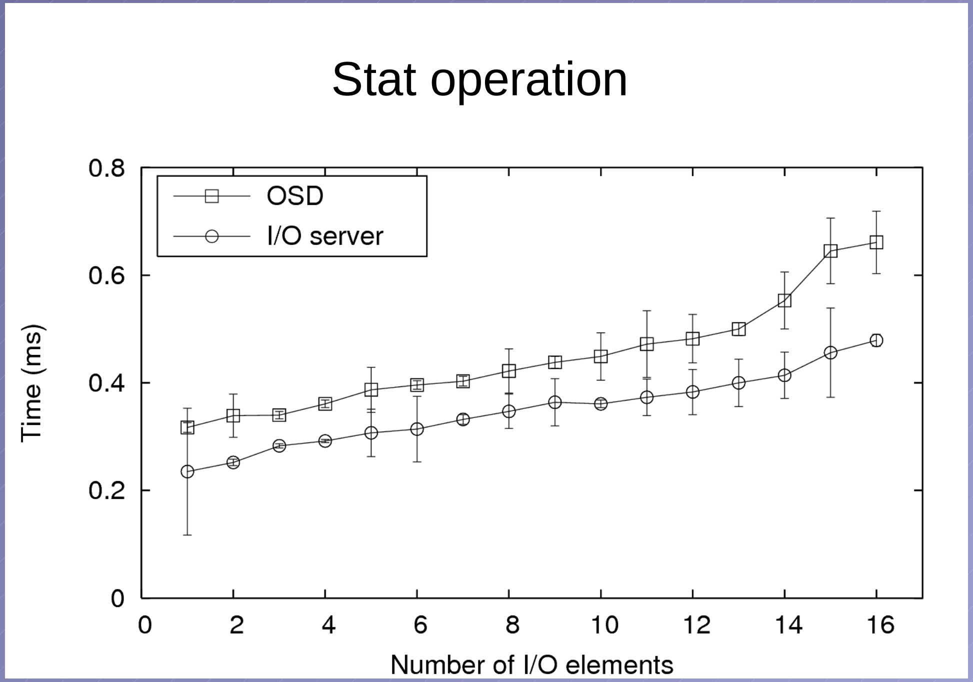
mdfile

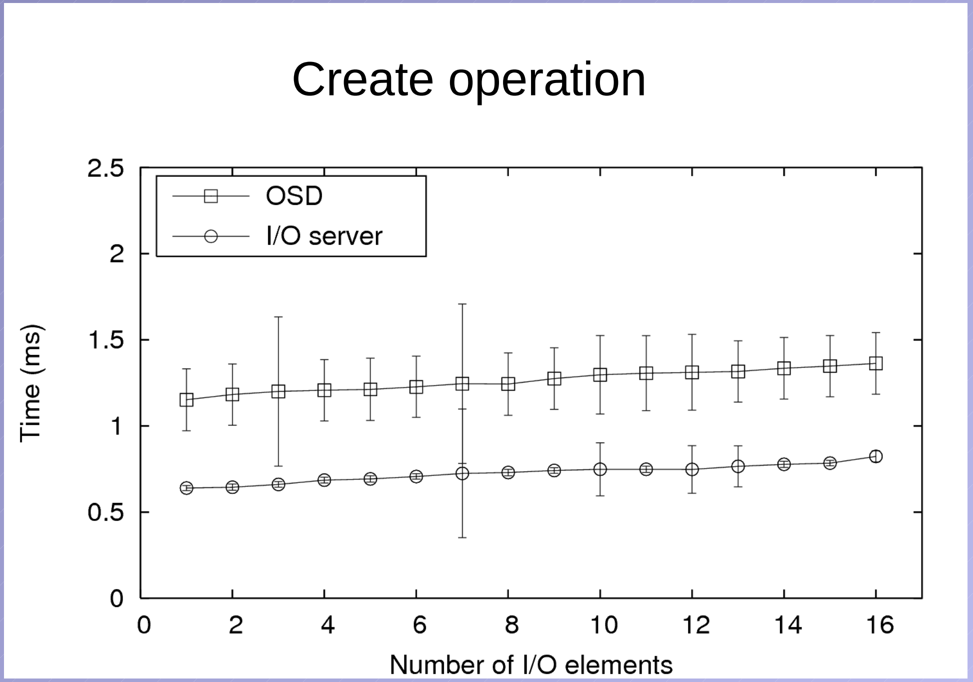
Overheads

Processing phase	Overhead
SQLite	$81.3 \pm 0.3\ \mu s$
CDB	$2.2 \pm 0.5\ \mu s$
iSCSI	$29.9 \pm 1.7\ \mu s$
Initiator	$125.6 \pm 2.9\ \mu s$
Total	$239.0 \pm 1.1\ \mu s$

- SQLite is handy, but slow
- iSCSI processing can be overlapped with threading

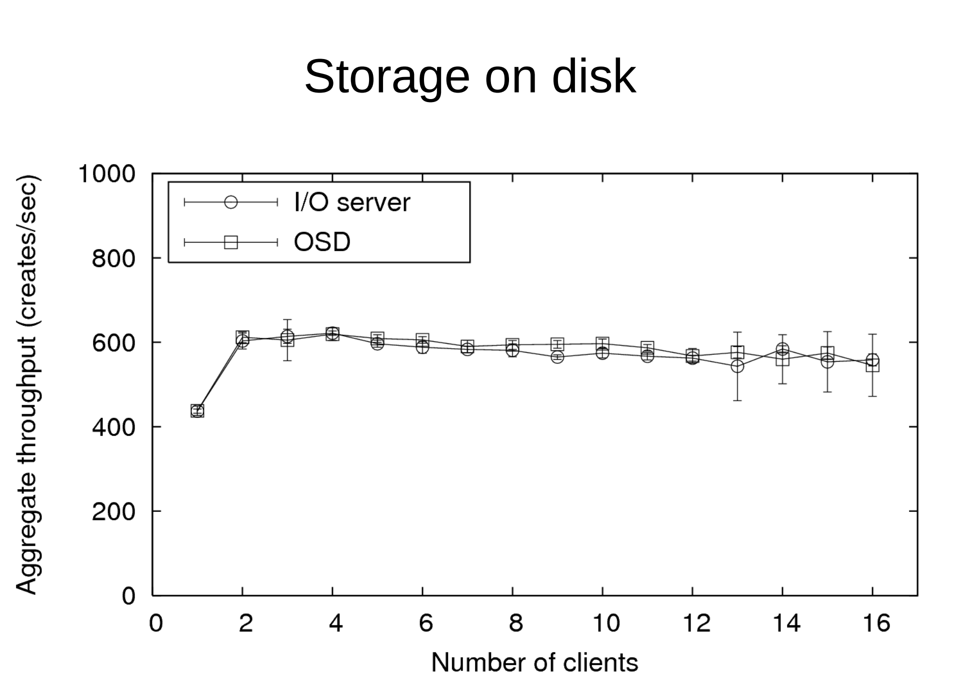
PVFS Latency

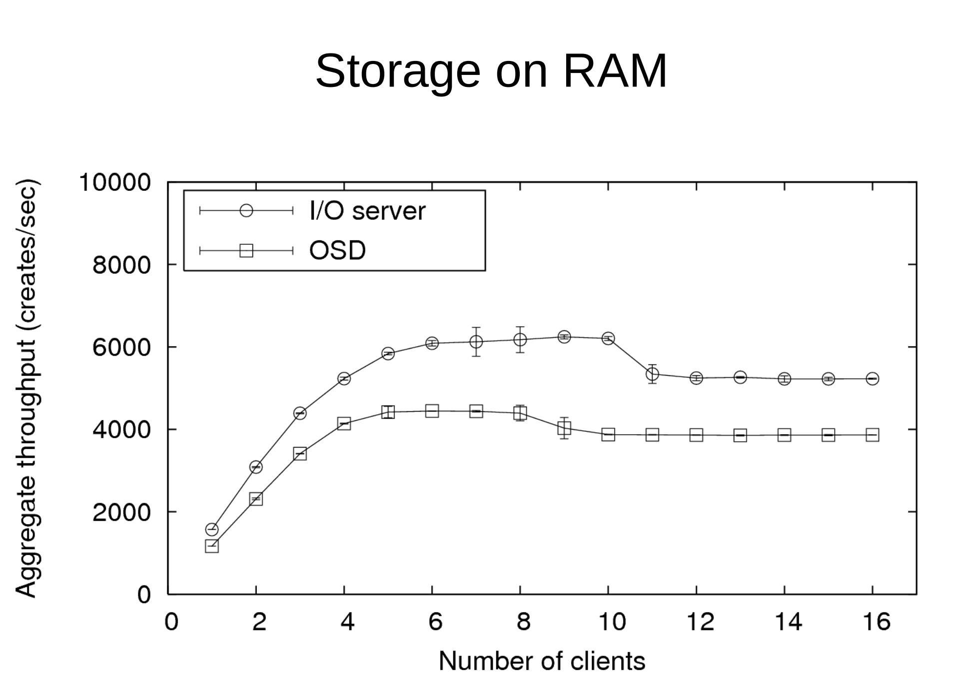




- OSD slower than PVFS, mostly due to 83µs SQLite
- Create slower but scales identically
- "Ping" operation slightly faster on OSD (not shown): avoids SQLite processing

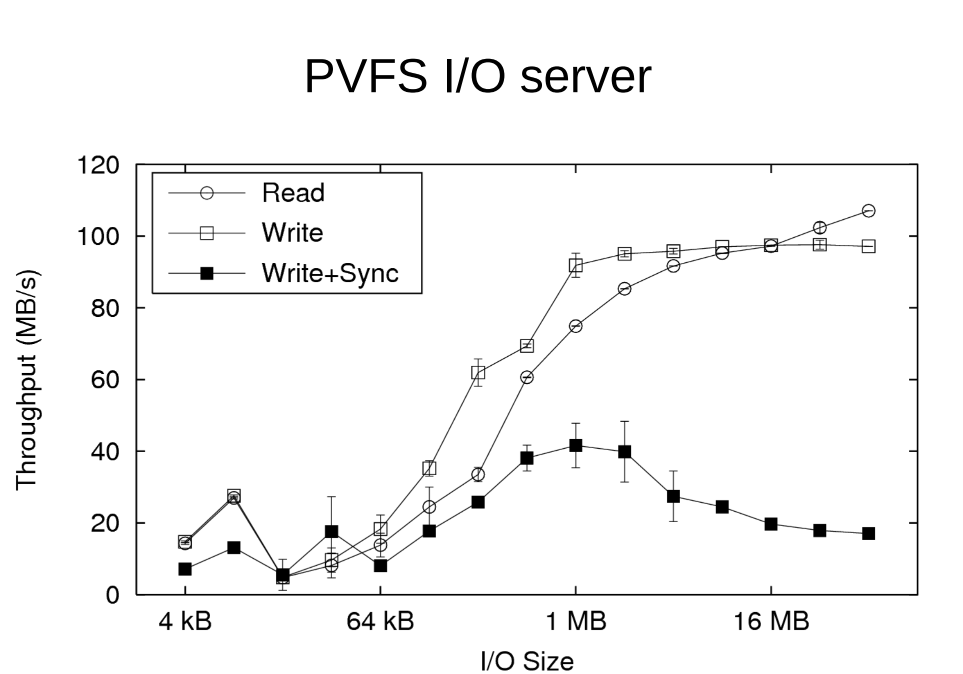
Create scaling with #clients

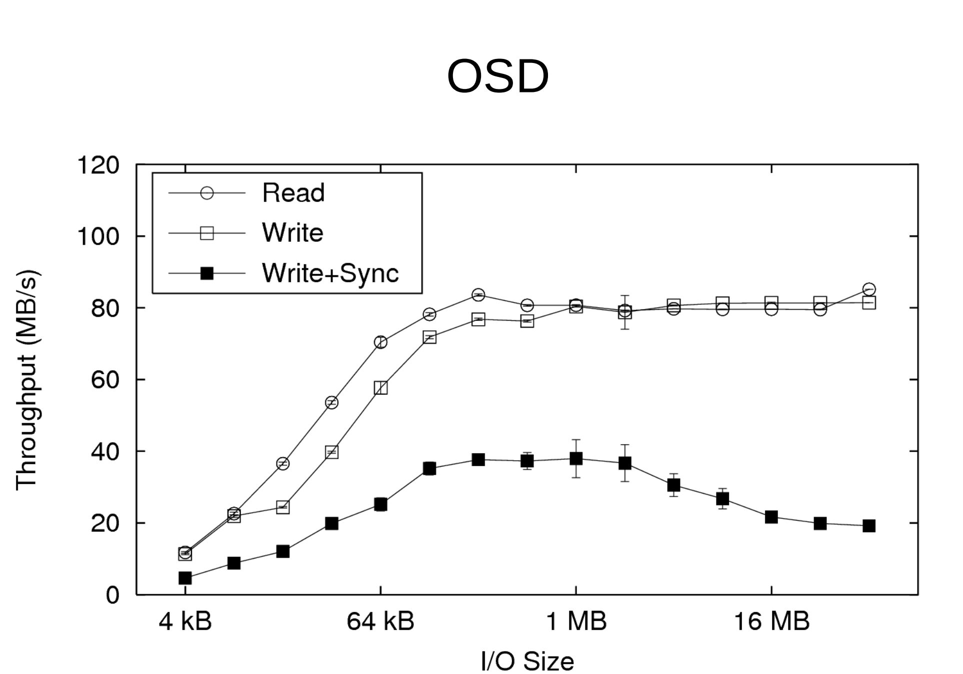




- Object database on disk: identical performance
- On RAM, OSD only 80% rate of PVFS I/O server
- Recent threading addition reduces this gap

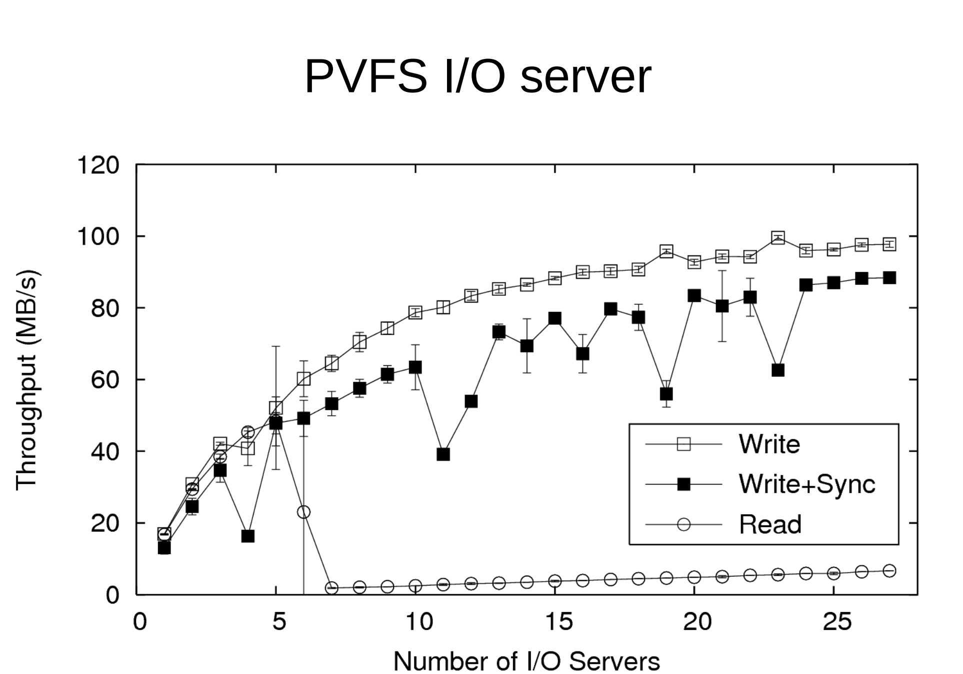
I/O Throughput

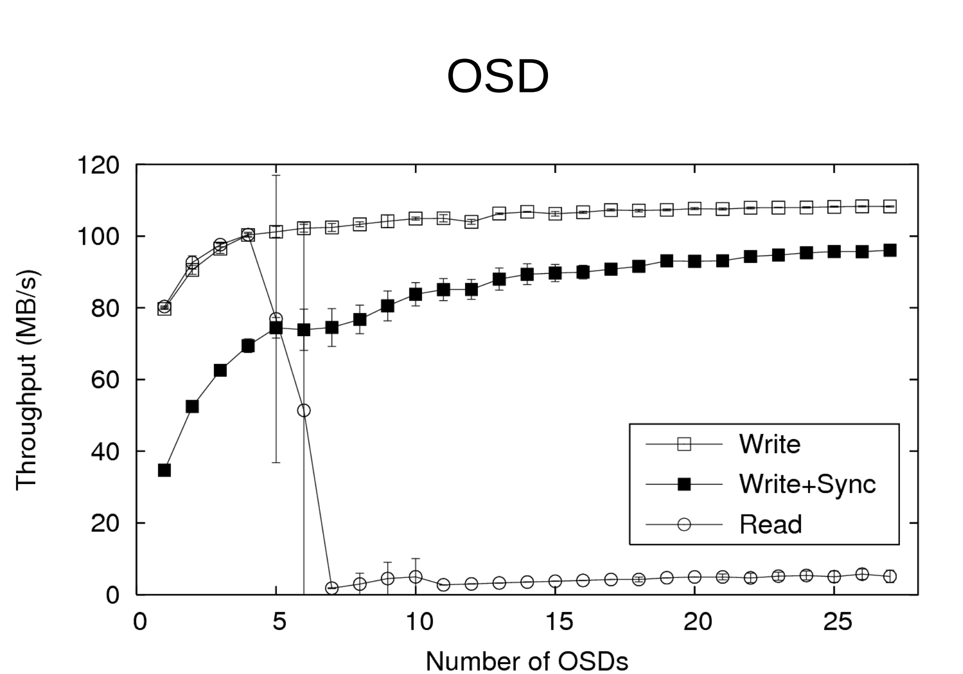




- One client writing or reading to one disk, using perf
- PVFS achieves higher throughput at large messages
- OSD ramps up better at small messages
  (PVFS flow tuning issue)

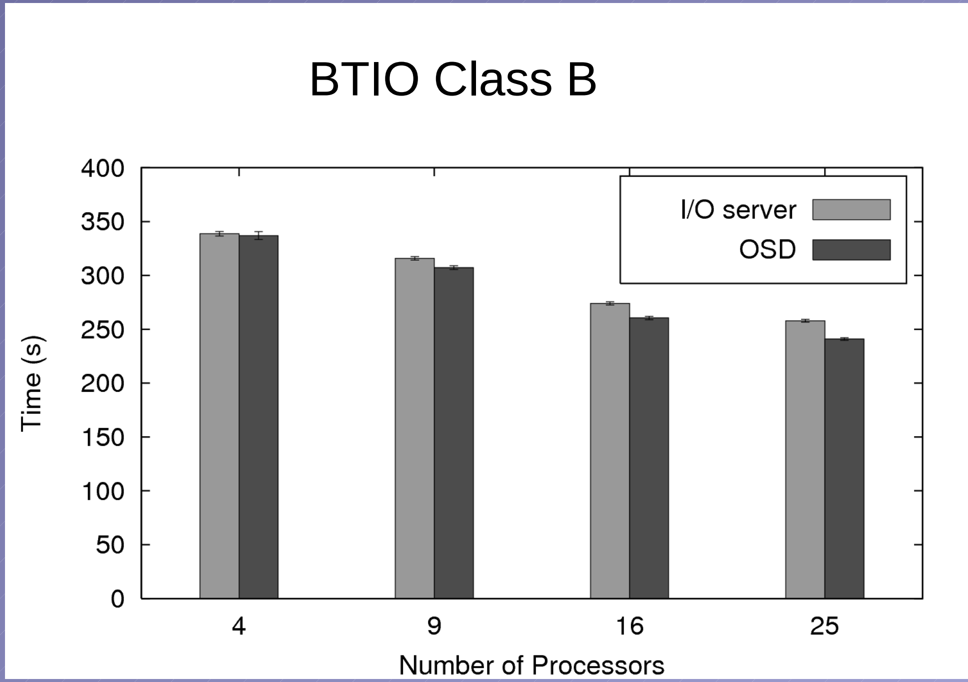
I/O Server Scaling

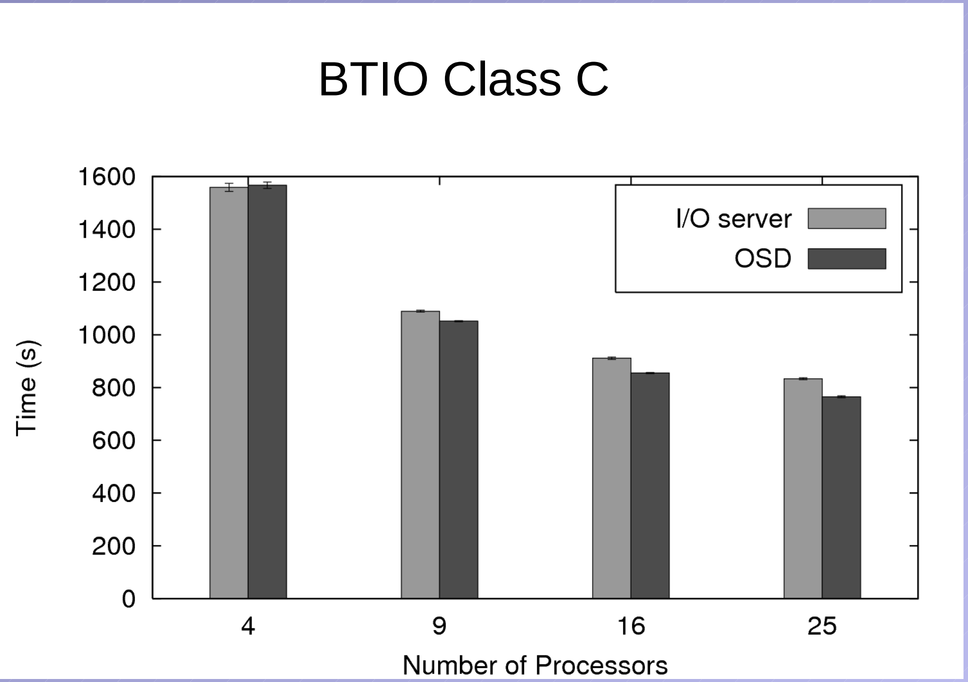




- One client, many I/O servers (or OSDs), perf at 64 kB
- Both ramp up okay
- PVFS penalized by default flow tunings and small size
- Terrible read behavior from TCP congestion control
- (iSER work removes this bottleneck.)

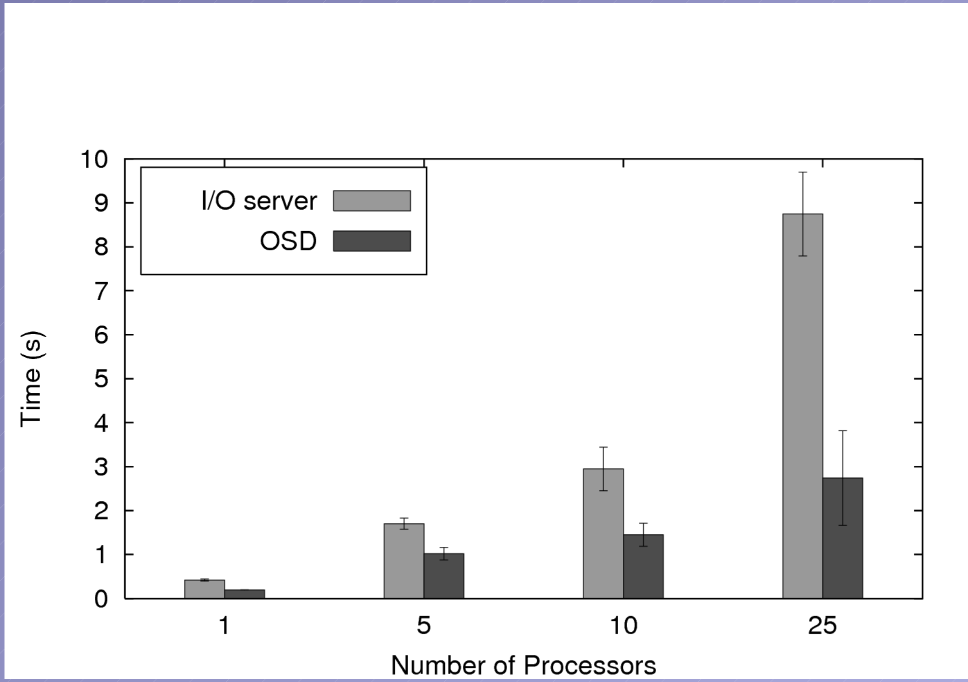
Application (BTIO) Performance





- Near identical performance
- Goal is to achieve similar performance, not improvement

Application (Flash) Performance



- Goal is to achieve similar performance, not improvement
- Stock PVFS performs poorly due to small message sizes

Major Accomplishments

- Use OSDs natively in a real parallel file system
- Infrastructure for OSDs
  - Library for OSD commands at initiator
  - Emulator for OSD target
  - iSCSI over RDMA for performance

- Revisit assumptions in PFS architecture
  - separate data and metadata
  - servers in the data path
  - servers at all

Barriers

- PVFS server functionality assumptions
  - handle mapping
  - server-initiated operations (soon)
- Other parallel FS assumptions
  - Ceph: servers forward write data to other servers
  - Lustre: byte-range locking, callbacks
  - Panasas: closer fit
  - NFSv4: OSD mapping exists (datafile only)
- lack of OSD hardware
- need for OSD extensions
  - atomic operations
  - scatter/gather on server
  - server-chosen retrieved attributes offset

Publications

- *Integrating parallel file systems with object-based storage devices*
  - SC07, Reno, NV, November 2007
  - system design, PFS mapping, PVFS impl.
  - overheads, metadata scaling, IO throughput, apps
- *Attribute storage design for object-based storage devices*
  - MSST07, San Diego, CA, September 2007
  - details on SQL use in metadata storage
- *iSER storage target for object-based storage devices*
  - MSST07 SNAPI Workshop, San Diego, Sept 07
  - iSCSI over RDMA
- Metadata design paper
  - to FAST08, due Sep 07

Staff

- Pete Wyckoff
  - PI, 25+% time
  - initiator and PVFS lead
- Ananth Devulapalli
  - OSC staff, 25+% time
  - target lead
- Dennis Dalessandro
  - OSC staff, 25% time
  - network lead

Students

- Nawab Ali
  - OSU graduate student, CS, 2nd year, half time
  - advisor: P. Sadayappan
  - parallel FS metadata design
- Paul Betts
  - OSU undergraduate, senior CS, 15 hr/wk
  - initiator kernel module, python interface
- Alex Moore
  - OSU undergraduate, senior CS, 15 hr/wk
  - target attribute handling
- Tim Arnold
  - OSU undergraduate, sophomore physics, 15 hr/wk
  - LIST, collection functions

Conclusion

- Enable higher semantic interface to storage
- Avoid middle-box interference on data path
- Understand roles of clients, metadata servers, IO servers, lock servers, etc in parallel FSes

- Actively sharing code with
  - Panasas
  - Seagate
  - Fujitsu
- Goal: build a single shared OSD environment to encourage research by ourselves and others